AMBIENT CALCULUS-BASED MODAL LOGIC MODEL CHECKING

RELATED APPLICATIONS

This application is related to the cofiled, copending and coassigned application entitled "Ambient Calculus-Based Modal Logics for Mobile Ambients" [docket no. 1018.021US1].

FIELD OF THE INVENTION

This invention relates generally to ambient calculus-based modal logics, and more specifically to model checking for such ambient calculus-based modal logics.

BACKGROUND OF THE INVENTION

Computing has become increasingly interconnected. Whereas before computers were discrete, unconnected units, because of the Internet as well as other networks, they are increasingly fluid, interconnected units. A computer program, which may be made up of one or more executable processes, or threads, may be mobile. For example, a thread of the program may move from computer to computer over the Internet. It may be executed in a distributed fashion over many computers, or a different instance of the thread may be run on each of many computers.

The movement of threads from computer to computer, or even to different parts within the same computer, poses new security and other risks for which there is no formal analysis mechanism. For example, a thread may be unstable, such that having it be run on a particular computer may cause the computer to crash. More so, the thread may be

malicious, such as part of a virus program, such that its purpose is to compromise the computers it moves to.

More specifically, there are two distinct areas of work in mobility: mobile computing, concerning computation that is carried out in mobile devices (laptops, personal digital assistants, etc.), and mobile computation, concerning mobile code that moves between devices (agents, etc.). Mobility requires more than the traditional notion of authorization to run or to access information in certain domains: it involves the authorization to enter or exit certain domains. In particular, as far as mobile computation is concerned, it is not realistic to imagine that an agent can migrate from any point A to any point B on the Internet. Rather an agent must first exit its administrative domain (obtaining permission to do so), enter someone else's administrative domain (again, obtaining permission to do so) and then enter a protected area of some machine where it is allowed to run (after obtaining permission to do so).

Access to information is controlled at many levels, thus multiple levels of authorization may be involved. Among these levels we have: local computer, local area network, regional area network, wide-area intranet and internet. Mobile programs should be equipped to navigate this hierarchy of administrative domain, at every step obtaining authorization to move further. Laptops should be authorized to access resources depending on their location in the administrative hierarchy.

In general, a process or thread resides within a container referred to as an ambient. The ambient includes one or more processes or threads, as well as any data, etc., that move with the processes or threads. An ambient that can move is referred to as a mobile ambient. The ambient can be any type of container: a software container such as a

particular part of an operating system, for example, as well as a hardware container, such as a particular computer or peripheral device.

More specifically, an ambient has the following main characteristics. First, an ambient is a bounded placed where computation happens. The interesting property here is the existence of a boundary around an ambient. Examples of ambients include: a web page (bounded by a file), a virtual address space (bounded by an addressing range), a Unix file system (bounded within a physical volume), a single data object (bounded by "self") and a laptop (bounded by its case and data ports). Non-examples are: threads (the boundary of what is "reachable" is difficult to determine) and logically related collections of objects.

Second, an ambient is something that can be nested within other ambients. For example, to move a running application from work to home, the application must be removed from an enclosing (work) ambient and inserted in a different enclosing (home) ambient. A laptop may need a removal pass to leave a workplace, and a government pass to leave or enter a country.

Third, an ambient is something that can be moved as a whole. If a laptop is connected to a different network, all the address spaces and file systems within it move accordingly and automatically. If an agent is moved from one computer to another, its local data should move accordingly and automatically.

As mentioned, there is no formal analysis mechanism within the prior art for such mobile ambients. This means that there is no manner by which to describe formally, for example, a security policy for a given computer system, which could be applied against a mobile ambient within a formal analysis mechanism to determine if the ambient poses a

security or other risk to the system. In particular, most formal analysis mechanisms, or frameworks, only provide for temporal distinction among processes and ambients, but assume that the processes and ambients are stationary – or otherwise do not provide for spatial distinction among them. Furthermore, there is no manner by which to formally verify that a policy or other model for process and ambients can be verified for correctness.

For these and other reasons, there is a need for the present invention.

SUMMARY OF THE INVENTION

The invention relates to ambient calculus-based modal logic model checking. In one embodiment, a computer-implemented method receives a process, which is also referred to as a thread or agent in varying embodiments. The method analyzes the process against a formula using a predetermined modal logic based on ambient calculus. The formula, for example, can represent a model to be checked, a policy to be verified, such as a security policy, etc. The method finally outputs whether the process satisfies the formula or not.

In one embodiment, analysis of the process against the formula is conducted in a recursive manner. The process is normalized to determine whether the process comprises only a single element. The process is partitioned to determine whether each component of the process satisfies the formula. A plurality of names of the process is determined, and it is verified that a name exists for the formula that is unequal to any of this plurality of names. Each sublocation of the process is analyzed against the formula. The spatial reach of the process is also analyzed against the formula.

Embodiments of the invention provide for advantages over the prior art. A policy, such as a security or mobility policy, expressed in terms of a formula according to the modal logic can be verified in a formal manner. For example, the logic can be used to describe a policy as how an applet can move around among different containers, or ambients. A process can then be matched, or analyzed, against this formal description of the policy. The policy can be intricate, stating, for example, how a process can run on a specific machine, in detail.

Embodiments of the invention include computer-implemented methods, computer-readable media, and computerized systems of varying scope. Still other embodiments, advantages and aspects of the invention will become apparent by reading the following detailed description, and by reference to the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of an operating environment in conjunction with which embodiments of the invention may be practiced;

FIG. 2 is a diagram of an example environment of ambients and processes in conjunction with which embodiments of the invention may be practiced;

FIG. 3 is a flowchart of a method according to an embodiment of the invention; and,

FIGs. 4-5 are diagrams of example situations of mobile ambients utilized in conjunction with the modal logic of varying embodiments of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Organization of the Detailed Description

The detailed description is organized as follows. The first section, the introduction, provides guidelines as to how to interpret the other sections of the detailed description. The second section describes an operating environment in context with which embodiments of the invention can be practiced. The third section provides a description of a mobile computing environment, which also gives guidance as to the context in which embodiments of the invention can be practiced. The fourth section describes modal logics, in accordance with which embodiments of the invention can be practiced. This fourth section includes various sub-sections, each of which detail different aspects of such modal logics. The fifth section highlights some examples of processes and formulas in the context of such modal logics.

The sixth section presents methods according to embodiments of the invention, which rely on the modal logics of the fourth section. The methods relate generally to analyzing processes against formulas in the context of the modal logics. Finally, a conclusion is given in the seventh section of the detailed description.

Introduction

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those

skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, electrical and other changes may be made without departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. (It is noted that the terms document and text are used interchangeably herein and should be construed as interchangeable as well.)

It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as processing or computing or calculating or determining or

displaying or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

Operating Environment

Referring to FIG. 1, a diagram of the hardware and operating environment in conjunction with which embodiments of the invention may be practiced is shown. The description of FIG. 1 is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment in conjunction with which the invention may be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc., that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PC's, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a

distributed computing environment, program modules may be located in both local and remote memory storage devices.

The exemplary hardware and operating environment of FIG. 1 for implementing the invention includes a general purpose computing device in the form of a computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that operatively couples various system components include the system memory to the processing unit 21. There may be only one or there may be more than one processing unit 21, such that the processor of computer 20 comprises a single central-processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment. The computer 20 may be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited.

The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory may also be referred to as simply the memory, and includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within the computer 20, such as during start-up, is stored in ROM 24. The computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media.

The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive

interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for the computer 20. It should be appreciated by those skilled in the art that any type of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24, or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor, computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 20 may operate in a networked environment using logical connections to one or more ~~remote~~ computers, such as remote computer 49. These logical connections are achieved by a communication device ~~coupled to or~~ a part of the

computer 20; the invention is not limited to a particular type of communications device. The remote computer 49 may be another computer, a server, a router, a network PC, a client, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local-area network (LAN) 51 and a wide-area network (WAN) 52. Such networking environments are commonplace in office networks, enterprise-wide computer networks, intranets and the Internal, which are all types of networks.

When used in a LAN-networking environment, the computer 20 is connected to the local network 51 through a network interface or adapter 53, which is one type of communications device. When used in a WAN-networking environment, the computer 20 typically includes a modem 54, a type of communications device, or any other type of communications device for establishing communications over the wide area network 52, such as the Internal. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It is appreciated that the network connections shown are exemplary and other means of and communications devices for establishing a communications link between the computers may be used.

Mobile Computing Environment

In this section of the detailed description, an example mobile computing environment in conjunction with which embodiments of the invention can be practiced.

That is, an example mobile computing environment, made up of ambients (containers) and processes (threads), is presented. Modal logics can then be used to represent these ambients and processes, as well as describe configurations of multiple such ambients and processes, and policies and formulas against which specific ambients and processes can be applied to determine if they satisfy the policies and formulas. That is, model checking as described herein can be used in accordance with such modal logics.

Referring to FIG. 2, an example mobile computing environment 200 is shown. The environment 200 specifically includes ambients, or containers, 202, 204 and 206. As shown in FIG. 2, the ambient 202 resides within the ambient 204. The ambient 202 is named *a*; the ambient 204 is named *b*; and, the ambient 206 is named *c*. A process `P` resides within the ambient 204, while a process `Q` resides within the ambient 202, and processes `R` and `S` reside within the ambient 206.

As has been described, each ambient, or container, can be a software or a hardware container. A software container may be a particular area defined by an operating system. Examples include stacks, heaps, sand boxes, as the latter term is referred to in the context of the Java programming language, etc. A hardware container may be a particular computer, such as a client or a server computer, as well as a particular computer peripheral. An example of a computer has been described in the preceding section of the detailed description.

More specifically, an ambient as used herein has the following properties:

- Each ambient has a name. The name of an ambient is used to control access (entry, exit, communication, etc.). In a realistic situation the true name of an ambient would be guarded very closely, and only specific capabilities would be handed out about how to use the name. In our examples we are usually more liberal in the handling of names, for sake of simplicity.

- Each ambient has a collection of local agents (referred interchangeably herein as threads or processes). These are the computations that run directly within the ambient and, in a sense, control the ambient. For example, they can instruct the ambient to move.
- Each ambient may have a collection of subambients. Each subambient has its own name, agents, subambients, etc.

Names refer to:

- something that can be created, passed around and used to name new ambients.
- something from which capabilities can be extracted.

The logic of embodiments of the invention pertains to a mobile computing environment. Thus, the ambients of FIG. 2 are mobile. As shown in FIG. 2, for example, the ambient 202 is moving out of the ambient 204. There may be, for example, a particular policy or formula, expressed in the logic, that defines whether such a move can occur, such that it can be applied against the ambient 202 and the policy therein to determine whether such a move should be allowed to occur. Each of the ambients and their resident processes are also representable in the logic of embodiments of the invention, which is described in the next section of the detailed description.

Modal Logic

In this section of the detailed description, modal logics based on ambient calculus, and providing for spatial relationships among processes of containers are presented. The logic makes assertions about the containment and contiguity of containers. Part of the logic is concerned with matching the syntactic structure of expressions in the calculus. The matching of the structure of formulas to the structure of processes is done in a flexible manner, up to a process equivalence, such that it is not entirely syntactical. A number of logical inference rules, including rules for propositional logic, rules for modal

operators such as time, space and validity, and rules for locations and process composition are also derived.

Basic Ambient Calculus

The following table summarizes a basic ambient calculus upon which a modal logic according to an embodiment of the invention is based. There is no name restriction in the basic ambient calculus. The subsequent tables summarize the syntax of processes, the structural congruence relation between processes, and the reduction semantics.

$P,Q,R ::=$	processes	
$\mathbf{0}$		inactivity
$P \mid Q$		composition
$!P$		replication
$M[P]$		ambient
$M.P$		capability action
$(n).P$		input action
$\langle M \rangle$		async output action
$M ::=$	messages	
n		name
in M		can enter into M
out M		can exit out of M
open M		can open M
ε		null
$M.M'$		path

Inactivity for a process means that the process does nothing; that is, it has no activity. The composition $P \mid Q$ means there is a resulting process composed of both P and Q. Replication means that the process has been replicated, or duplicated, as opposed to moving from one container to another; the replication $!P$ means the same effectively as an infinite array of replicas of P running in parallel.. The ambient $M[P]$ means that

the process P resides within the container, or ambient, M. The capability action $M.P$ means that the process is capable of the action, or functionality, M followed by the continuation P. The input action $(n).P$ means that the process can accept an input message, bind it to n and continue with P. The asynchronous output action $\langle M \rangle$ means that the process performs an output of the message M and stops.

A message expression M can take one of several forms. It can be a name n. It can be one of the capabilities, *in M*, *out M*, or *open M*, whose effect when exercised, respectively, is to move the enclosing ambient into a sibling M, to move the enclosing ambient out its parent M, or to dissolve the boundary around an adjacent ambient M. It can be a null capability ε. Or it may be a path $M.M'$, whose effect is that of exercising first M and then M'. A process P has a set of free names, written as *fn(P)*, which generally refers to any of the names textually occurring in the process P can take. More formally, *fn(P)* is defined by the following table.

(1) $fn(\mathbf{0}) @ \phi$	(8) $fn(n) @ \{n\}$
(2) $fn(P \mid Q) @ fn(P) \cup fn(Q)$	(9) $fn(in\ M) @ fn(M)$
(3) $fn(!P) @ fn(P)$	(10) $fn(out\ M) @ fn(M)$
(4) $fn(M[P]) @ fn(M) \cup fn(P)$	(11) $fn(open\ M) @ fn(M)$
(5) $fn(M.P) @ fn(M) \cup fn(P)$	(12) $fn(\varepsilon) @ \phi$
(6) $fn((n).P) @ fn(P) - \{n\}$	(13) $fn(M.M') @ fn(M) \cup fn(M')$
(7) $fn(\langle M \rangle) @ fn(M)$	

The thirteen statements within this table are explained as follows. The first statement states that there are no free names for the inactivity process. The symbol @ specifies that the left-hand side of the symbol is defined as the right-hand side of the symbol. This definition is applicable in any statement in which the symbol @ appears.

The second statement states that the free names for the composition $P|Q$ are the free names for P conjoined with the free names for Q. The third statement states that when a process is replicated from another process, it has the same free names as that latter process. The fourth statement states that the free names of a container M having therein a process P are the free names of M by itself conjoined with the free names of P – that is, $M[P]$ cannot take on any names that are not allowed by either M itself or P itself. The fifth statement states that the free names of the input action $(n).P$ are the free names of the process P, minus the name n.

The seventh statement states that the free names of the asynchronous output action $\langle M \rangle$ are the same as the free names of the message M itself. The eighth statement means that the free names of a name n is the singleton set containing n. The ninth statement means that the free names of the capability "can enter into M" are the same as the free names of M itself. Likewise, the tenth and eleventh statements means that the free names of the capabilities "can exit out of M" and "can open M," respectively, are the same as the free names of M itself. The twelfth statement states that there are no free names for the null capability. The last statement states that the free names of the path $M.M'$ are equal to the free names of M conjoined with the free names of M'.

Furthermore, it is noted that the terminology $P\{n \leftarrow M\}$ is used for the substitution of the capability M for each free occurrence of the name n in the process P, and similarly for $M\{n \leftarrow M'\}$.

Structural congruence is defined as summarized in the following table. We use the symbol $\equiv$ to denote the relation of structural congruence, and in general write the phrase $P \equiv Q$ to mean that processes P and Q are equal up to structural congruence.

$(1)\, P \equiv P$	(Struct Refl)
$(2)\, P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$(3)\, P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$(4)\, P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$(5)\, P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$(6)\, P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Struct Amb)
$(7)\, P \equiv Q \Rightarrow M.P \equiv M.Q$	(Struct Action)
$(8)\, P \equiv Q \Rightarrow (x).P \equiv (x).Q$	(Struct Input)
$(9)\, \varepsilon.P \equiv P$	(Struct ε)
$(10)\, (M.M').P \equiv M.M'.P$	(Struct .)
$(11)\, P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(12)\, (P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$(13)\, !P \equiv P \mid !P$	(Struct Repl Par)
$(14)\, P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(15)\, !\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Repl)

This table is explained as follows. Structural reflectivity means that P is equal to P. Structural symmetry means that if P equals Q, then Q equals P. Structural transitivity means that if P equals Q and Q equals R, then P also equals R. The fourth statement means that if P equals Q, then the composition $P \mid R$ is equal to the composition $Q \mid R$. The fifth statement means that if P equals Q, then the replication of P equals the replication of Q. The sixth statement means that if P equals Q the ambient M in which P is contained, $M[P]$, equals the ambient M in which Q is contained, $M[Q]$. Similarly, the seventh statement means that if P equals Q, then the exercise of the expression M before the action of P, $M.P$, is equal to the exercise of the expression M before the action of Q,

$M.Q$. The eighth statement means that if P equals Q, then P prefixed by the input action x is equal to Q prefixed by the input action x.

The ninth statement means that prefixing the process P with the null capability is the same as just stating the process P. The tenth statement means that stating $(M.M').P$ is the same as stating $M.M'.P$. The eleventh statement is the commutative property, that the composition $P|Q$ is equal to the composition $Q|P$. The twelfth statement is the associative property, that the composition of $(P|Q)$ and R is equal to the composition of P and $(Q|R)$. The thirteenth statement states that the replication of P is equal to the composition $P|!P$. The fourteenth statement is an identity statement, that the composition of P and the inactivity process is equal to P, while the fifteenth statement states that replicating the inactivity process is equal to the inactivity process itself.

Reduction is summarized in the next table. In it, the left side of the arrow ($\rightarrow$) reduces to the expression on the right side of the arrow.

$(1)\, n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$	(Red In)
$(2)\, m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$(3)\, open\ n.P \mid n[Q] \rightarrow P \mid Q$	(Red Open)
$(4)\, (n).P \mid (M) \rightarrow P\{n \leftarrow M\}$	(Red Comm)
$(5)\, P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$	(Red Amb)
$(6)\, P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$(7)\, P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(Red $\equiv$)
$(8) \rightarrow *$	reflexive and transitive closure of $\rightarrow$

Finally, the following syntactic conventions and abbreviations, as summarized in the next table, are used herein. A fact is also provided.

Syntactic conventions

$!P \mid Q$	is read	$(!P) \mid Q$
$M.P \mid Q$	is read	$(M.P) \mid Q$
$(n).P \mid Q$	is read	$((n).P) \mid Q$

Abbreviations

$n[\ ]$	$@n[\mathbf{0}]$
M	$@M.\mathbf{0}$ (where appropriate)

Fact

$n[P] \equiv m[P']$ iff $n = m$ and $P \equiv P''$

Logical Formulas

In this next sub-section, logical formulas of the modal logic, according to one embodiment of the invention, are presented. The logical formulas are based on a modal predicate logic with classical negation, as can be appreciated by those of ordinary skill within the art. Many connectives are interdefinable: existential formulations are given preference, because they have a more intuitive meaning than the corresponding universal ones. Two tables are provided: one specifying the logical formulas, and the next specifying connectives derived from the logical formulas.

A, B, C ::=		
1	**T**	true
2	¬A	negation
3	A ∨ B	disjunction
4	*n*[A]	location
5	A' | A"	composition
6	∃*n*.A	existential quantification over names
7	✧A	somewhere modality (spatial)
8	◊A	sometime modality (temporal)
9	A@*n*	location adjunct
10	A>B	composition adjunct

The logical formulas of the preceding table are described as follows. The first statement is a logical true, while the second statement is a logical negation and the fourth statement is a logical disjunction. The fourth statement means that the process A is located within the container, or ambient, *n*. The fifth statement is a logical composition. The sixth statement specifies the existential quantifier operation, that there is some process A within the container named *n*. The seventh statement specifies a spatial operator, that somewhere, at some location, the process A exists. That is, within some container, anywhere in the domain space being considered, the process A exists. Similarly, the eighth statement specifies a temporal operator, that at some point in time, the process A will exist (or currently exists). The ninth statement specifies that the process A exists within the container named *n*. Finally, the tenth statement is a logical composition adjunct.

1 **F**	@¬**T**	false
2 A ∧ B	@¬(¬A ∨ ¬B)	conjunction
3 A ⇒ B	@¬A ∨ B	implication
4 A ⇔ B	@(A ⇒ B) ∧ (B ⇒ A)	logical equivalence
5 A B	@¬(¬A | ¬B)	decomposition
6 !A	@A **F**	every component satisfies A
7 ?A	@A | **T** (⇔ ¬!¬A)	some component satisfies A
8 ∀*n*.A	@¬∃*n*.¬A	universal quantification over names
9 ¤A	@¬✧¬A	everywhere modality (spatial)
10 □A	@¬◊¬A	everytime modality (temporal)
11 A@	@∀*n*.A@*n*	in every location context
12 >A	@**I**>A	in every composition context

The derived connectives of the preceding table are explained as follows. The first statement is the logical false, and is derived and defined as a function of the logical true.

The second statement is the logical conjunction, while the third statement is the logical implication and the fourth logical equivalence. The fifth statement specifies logical decomposition. The sixth statement defines !A as universal satisfaction, that every component satisfies the process A. Likewise, the seventh statement defines ? A as partial satisfaction, that some component satisfies the process A. The eighth statement defines the universal quantifier ∀ in terms of the existential quantifier ∃; that all the processes A are within the container *n*. The ninth statement states that the process A exists everywhere, from a spatial perspective, while the tenth statement states that the process A has existed, and still exists, at every time. The eleventh and twelfth statements specify the in every location context and the in every composition context, respectively, and are derived from the eleventh and twelfth logical formula statements of the logical formulas table.

Finally, the following syntactic conventions are utilized herein.

- Parentheses are used for explicit precedence.
- Infix ~> binds stronger than |, and they both bind stronger than the standard logical connectives.
- Standard precedence is used for the standard logical connectives.
- Quantifiers and modalities extend to the right as much as possible.

Satisfaction

The satisfaction relation *P* A (process *P* satisfies formula A) is defined inductively in the following tables, where Π is the sort of processes, Φ is the sort of formulas, and Λ is the sort of names. Quantification and sorting of meta-variables are

made explicit because of subtle scoping issues, particularly in the definition of $P\ \exists n.\mathtt{A}$. Similar syntax for logical connectives is used at the meta-level and object-level.

The meaning of the temporal modality is given by reductions in the operational semantics of the ambient calculus. For the spatial modality, the following definitions are needed. The relation $P{\downarrow}P'$ indicates that P contains P' within exactly one level of nesting. Then, $P{\downarrow}{*}P'$ is the reflexive and transitive closure of the previous relation, indicating that P contains P' at some nesting level. Note that P' constitutes the entire contents of an enclosed ambient.

$P{\downarrow}P'$ iff $\exists n, P''.\ P \equiv n[P'] \mid P''$

$\downarrow *$ is the reflexive and transitive closure of $\downarrow$

$\forall P{:}\Pi.$	$P\ \mathbf{T}$	$\cong$	
$\forall P{:}\Pi,\mathtt{A}{:}\Phi$	$P\ \neg\mathtt{A}$	$\cong$	$\neg P\ \mathtt{A}$
$\forall P{:}\Pi,\mathtt{A},\mathtt{B}{:}\Phi.$	$P\ \mathtt{A}\vee\mathtt{B}$	$\cong$	$P\ \mathtt{A}\vee P\ \mathtt{B}$
$\forall P{:}\Pi,n{:}\Lambda,\ \mathtt{A}{:}\Phi.$	$P\ n[\mathtt{A}]$	$\cong$	$\exists P'{:}\Pi.\ P \equiv n[P'] \wedge P'\ \mathtt{A}$
$\forall P{:}\Pi,\mathtt{A},\mathtt{B}{:}\Phi.$	$P\ \mathtt{A}\ \vert\ \mathtt{B}$	$\cong$	$\exists P',P''{:}\Pi.\ P \equiv P'\vert P'' \wedge P'\ \mathtt{A} \wedge P''\ \mathtt{B}$
$\forall P{:}\Pi,\ n{:}\Lambda,\ \mathtt{A}{:}\Phi.$	$P\ \exists n.\mathtt{A}$	$\cong$	$\exists m{:}\Lambda.\ P\ \mathtt{A}\{n{\leftarrow}m\}$
$\forall P{:}\Pi,\ \mathtt{A}{:}\Phi$	$P\ \Diamond\mathtt{A}$	$\cong$	$\exists P'{:}\Pi.\ P{\downarrow}{*}P' \wedge P'\ \mathtt{A}$
$\forall P{:}\Pi,\ \mathtt{A}{:}\Phi$	$P\ \Box\mathtt{A}$	$\cong$	$\exists P'{:}\Pi.\ P{\rightarrow}{*}P' \wedge P'\ \mathtt{A}$
$\forall P{:}\Pi,\ \mathtt{A}{:}\Phi$	$P\ \mathtt{A}@n$	$\cong$	$n[P]\ \mathtt{A}$
$\forall P{:}\Pi,\ \mathtt{A},\mathtt{B}{:}\Phi.$	$P\ \mathtt{A}{>}\mathtt{B}$	$\cong$	$\forall P'{:}\Pi.\ P'\ \mathtt{A} \Rightarrow P\vert P'\ \mathtt{B}$

The logical connectives of the preceding table are read as follows:

- Any process satisfies the **T** formula.
- A process satisfies the $\neg\mathtt{A}$ formula if it does not satisfy the A formula.
- A process satisfies the $\mathtt{A}\vee\mathtt{B}$ formula if it satisfies either the A or the B formula.
- A process P satisfies the $n[\mathtt{A}]$ formula if there exists a process P' such that $P \equiv n[P']$ and P' A.
- A process P satisfies the $\mathtt{A}'\mid\mathtt{A}''$ formula if there exist processes P' and P'' such that $P \equiv P' \mid P''$ with P' satisfying $\mathtt{A}'$ and P'' satisfying $\mathtt{A}''$.

- A process P satisfies the formula $\exists n.\mathtt{A}$ if there is a name m such that P satisfies $\mathtt{A}\{n \leftarrow m\}$. (N.B.: the meta-theoretical definition above precisely captures the fact that m can be instantiated to, but cannot itself clash with any name free in P.)
- A process P satisfies the formula $\Diamond$ A if A holds at some location P' within P, where "sublocation" is defined by $P \downarrow^* P'$.
- A process P satisfies the formula $\Diamond$A if A holds in the future for some residual P' of P, where "residual" is defined by $P * P'$.
- A process P satisfies the formula A@n if, when placed in an ambient n, the combination $n[P]$ satisfies A.
- A process P satisfies the formula A >B if, given any parallel context P' satisfying A, the combination $P' \mid P$ satisfies B. Another reading of P A >B is that P manages to satisfy B under any possible attack by an opponent that is bound to satisfy A. Moreover, " P satisfies ($\Box$A) >($\Box$A)" means that P preserves the invariant A.

$\forall P{:}\Pi.$	$\neg P$ **F**	
$\forall P{:}\Pi,$ A, B: $\Phi.$	P A$\wedge$B	iff P A $\wedge$ P B
$\forall P{:}\Pi,$ A, B: $\Phi.$	P A$\Rightarrow$B	iff P A $\Rightarrow$ P B
$\forall P{:}\Pi,$ A, B: $\Phi.$	P A$\Leftrightarrow$B	iff P A $\Leftrightarrow$ P B
$\forall P{:}\Pi,$ A, B: $\Phi.$	P A B	iff $\forall P', P''{:}\Pi.\ P \equiv P'\mid P'' \Rightarrow P'$ A $\vee$ P'' B
$\forall P{:}\Pi,$ A:$\Phi.$	P !A	iff $\forall P', P''{:}\Pi.\ P \equiv P'\mid P'' \Rightarrow P'$ A
$\forall P{:}\Pi,$ A: $\Phi.$	P ?A	iff $\exists P', P''{:}\Pi.\ P \equiv P'\mid P'' \wedge P'$ A
$\forall P{:}\Pi, n{:}\Lambda,$ A: $\Phi.$	P $\forall n$.A	iff $\forall m{:}\Lambda.\ P$ A$\{n \leftarrow m\}$
$\forall P{:}\Pi,$ A: $\Phi.$	P $\Box\!\!\!\!-$A	iff $\forall P'{:}\Pi.\ P \downarrow^* P' \Rightarrow P'$ A
$\forall P{:}\Pi,$ A: $\Phi.$	P $\Box$A	iff $\forall P'{:}\Pi.\ P \rightarrow^* P' \Rightarrow P'$ A
$\forall P{:}\Pi,$ A: $\Phi.$	P A@	iff $\forall n{:}\Lambda.\ P$ A@n
$\forall P{:}\Pi,$ A: $\Phi.$	P >A	iff $\forall P'{:}\Pi.\ P \mid P'$ A
$\forall P{:}\Pi,$ A, B: $\Phi.$	P >(A$\Rightarrow$B)	iff $\forall P'{:}\Pi.\ P' \mid P$ A $\Rightarrow$ $P' \mid P$ B *(cf. P A>B)*

The derived logical connectives of the preceding table are read as follows:

- No process satisfies the **F** formula.
- A process satisfies the A $\wedge$B formula if it satisfies both the A and the B formula.
- A process satisfies the A $\Rightarrow$B formula if either it does not satisfy the A formula or it satisfies the B formula.
- A process satisfies the A $\Leftrightarrow$B formula if it satisfies neither or both the A and B formulas.
- A process P satisfies the A'A" formula if for every decomposition of P into processes P' and P'' such that $P \equiv P'\mid P''$, either P' satisfies A' or P'' satisfies A".

- A process P satisfies the !A formula if every parallel component P' of P (such that $P \equiv P'|P''$, including $P' = \mathbf{0}$) satisfies the A formula.
- A process P satisfies the ?A formula if there is a parallel component P' of P (such that $P \equiv P'|P''$) that satisfies the A formula.
- A process P satisfies the formula $\forall n.$A if for every name m, P satisfies A$\{n \leftarrow m\}$.
- A process P satisfies the formula ¤A if A holds at every location P' within P, where "sublocation" is defined by $P{\downarrow}{*}P'$.
- A process P satisfies the formula □A if A holds in the future for every residual P' of P, where "residual" is defined by $P{\rightarrow}{*}P'$.
- A process P satisfies the formula A @ if, when placed in any ambient n, the combination $n[P]$ satisfies A.
- A process P satisfies the formula >A if for every process (i.e., for every context) the combination of P and with that process satisfies A.
- If process P satisfies the formula A>B, it means that in every context that satisfies A, the combination (of P and the context) satisfies B. Instead, if process P satisfies the formula >(A⇒B), it means that in every context, if the combination satisfies A then the combination satisfies B.

The following proposition states that the satisfaction relation is invariant under structural congruence.

~~$P \equiv P' \Rightarrow (P$ A $\Rightarrow P'$ A)~~

A list of examples of the satisfaction relations is now provided. These examples should appear intuitively true from the definitions.

Location

$n[]$ $n[\mathbf{T}]$

$n[] \mid \mathbf{0}$ $n[\mathbf{T}]$, because $n[] \mid \mathbf{0} \equiv n[]$

$n[m[]]$ $n[m[\mathbf{T}]]$

$\neg\mathbf{0}$ $n[\mathbf{T}]$

$\neg n[]$ $m[\mathbf{T}]$, if $n \neq m$

Composition

$n[] \mid m[]$ $n[\mathbf{T}] \mid m[\mathbf{T}]$

$n[] \mid m[]$ $m[\mathbf{T}] \mid n[\mathbf{T}]$, because $n[] \mid m[] \equiv m[] \mid n[]$

$n[] \mid P$ $n[\mathbf{T}] \mid \mathbf{T}$

$n[]$ $n[\mathbf{T}] \mid \mathbf{T}$, because $n[] \equiv n[] \mid \mathbf{0}$

$!n[]\ \ n[\mathbf{T}] \mid \mathbf{T}$, because $!n[] \equiv n[] \mid !n[]$

$\neg n[]\ \ n[\mathbf{T}] \mid n[\mathbf{T}]$

$\neg n[] \mid n[]\ \ n[\mathbf{T}]$

$\neg !n[]\ \ n[\mathbf{T}]$

$\neg n[] \mid \text{open } m\ \ n[\mathbf{T}]$

Quantification

$n[]\ \exists m.m[\mathbf{T}]$ iff $\exists p.\ n[]\ p[\mathbf{T}]$ iff $n[]\ n[\mathbf{T}]$ iff **true**

$n[m[]]\ \exists n.n[n[\mathbf{T}]]$ iff $\exists p.\ n[m[]]\ p[p[\mathbf{T}]]$ iff **false**

$\mathbf{0}\ \forall n.\ \neg n[\mathbf{T}]$

Spatial Modality

$n[m[]]\ \diamond m[\mathbf{T}]$

$\neg\ n[m[] \mid m[]]\ \diamond m[\mathbf{T}]$

Temporal Modality

$n[m[]] \mid \textit{open } n\ \ \Diamond m[\mathbf{T}]$

$n[n[]] \mid \textit{open } n\ \ \Box(n[\mathbf{T}] \mid \mathbf{T})$

Location Adjunct

$n[]\ \ m[n[\mathbf{T}]]@m$

$n[\textit{out } m]\ \ (\Diamond n[\mathbf{T}])@m$

Composition Adjunct

$n[]\ \ m[\mathbf{T}]>(n[\mathbf{T}] \mid m[\mathbf{T}])$

$\textit{open } n.\ m[]\ \ (\Box n[\mathbf{T}])>(\Diamond m[\mathbf{T}])$

Presence

$\textit{an } n \cong n[\mathbf{T}] \mid \mathbf{T}$	(there is now an n here)
$\textit{no } n \cong \neg \textit{an } n$	(there is now no n here)
$\textit{one } n \cong n[\mathbf{T}] \mid \textit{no } n$	(there is now exactly one n here)
$\textit{unique } n \cong n[\Bumpeq \textit{no } n] \mid \Bumpeq \textit{no } n$	(there is now exactly one n, and it is here)
$!(n[\mathbf{T}] \Rightarrow n[\mathrm{A}])$	(every n here satisfies A)

Validity and Satisfiability

It is noted that a formula is valid if it is satisfied by every process, and is satisfiable if it is satisfied by some process. This is summarized in the following table.

vld A	$\cong$	$\forall P{:}\Pi.\ P$ A	A is valid
sat A	$\cong$	$\exists P{:}\Pi.\ P$ A	A is satisfiable

From these definitions, the following are obtained:

$$\textit{vld}\ A \Rightarrow \textit{sat}\ A$$
$$\textit{vld}\ A \Leftrightarrow \neg\textit{sat}\ \neg A$$
$$\textit{vld}\ (A \wedge B) \Leftrightarrow \textit{vld}\ A \wedge \textit{vld}\ B$$
$$\textit{vld}\ (A \vee B) \Leftrightarrow \textit{vld}\ A \vee \textit{vld}\ B$$

Validity is used for modeling logical inference rules, as described in the next definition. A linearized notation is used for inference rules, where the usual horizontal bar separating antecedencts from consequents is written '/', and ';' is used to separate antecedents.

Definition (Sequents and Rules)

~~Sequents~~:

~~$A \quad B \cong \textit{vld}\ (A \Rightarrow B)$~~

Rules:

$$A_1 \quad B_1; \ldots; A_n \quad B_n \ / \ A \quad B \cong$$
$$A_1 \quad B_1 \wedge \ldots \wedge A_n \quad B_n \Rightarrow A \quad B \quad (n \geq 0)$$
$$A_1 \quad B_1 \,//\, A_2 \quad B_2 \cong$$
$$A_1 \quad B_1 / A_2 \quad B_2 \wedge A_2 \quad B_2 / A_1 \quad B_1$$

Inference Rules

In this section, logical inference rules from the satisfaction relation are derived.

The following is a non-standard presentation of the sequent calculus, where each sequent has exactly one assumption and one conclusion: $A \quad B$. This presentation is adopted because the logical connectives introduced later do not preserve the shape of multiple-assumption multiple-conclusion sequents. Moreover, in this presentation the rules of propositional logic become extremely symmetrical. Propositional logic is summarized in the following table.

~~(A-L) $A \wedge (C \wedge D) \quad B \,//\, (A \wedge C) \wedge D \quad B$~~

(A-R)	A (C∨D) ∨B // A C∨(D∨B)
(X-L)	A∧C B / C∧A B
(X-R)	A C∨B / A B∨C
(C-L)	A∧A B / A B
(C-R)	A B∨B / A B
(W-L)	A B / A∧C B
(W-R)	A B / A C∨B
(Id)	/ A A
(Cut)	A C∨B; A' ∧C B' / A∧A' B∨B'
(T)	A∧**T** B / A B
(F)	A **F**∨B / A B
(¬-L)	A C∨B / A∧¬C B
(¬-R)	A∧C B / A ¬C∨B
(∧)	A B; A' B' / A∧A' B∧B'
(∨)	A B; A' B' / A∨A' B∨B'

The standard deduction rules of propositional logic, both for the sequent calculus and for natural deduction, are derivable from the rules of the preceding table, as can be appreciated by those of ordinary skill within the art. As usual, A ⇒ B can be defined as ¬A∨B.

For predicate logic the syntax of formulas (but not of processes) is enriched with variables ranging over names. These variables are indicated by letters *x, y, z*. Quantifiers bind variables, not names. Then, if $fv(A)=\{x_1, \ldots, x_k\}$ are the free variables of A and $\varphi \in fv(A) \to \Lambda$ is a substitution of variables for names, A_φ for A $\{x_1 \leftarrow \varphi(x_1), \ldots, x_k \leftarrow \varphi(x_k)\}$ is written, and the following is defined:

$$\textbf{\textit{vld}}\ A \ \cong\ \forall P\colon \Pi.\ P\ A_\varphi$$

The following table summarizes quantifiers over names.

(∀-L)	A $\{x \leftarrow m\}$B / ∀x. A B	
(∀-R)	A B / A ∀x.B	Where $x \notin fv(A)$
(∃-L)	A B / ∃x. A B	Where $x \notin fv(B)$
(∃-R)	A B $\{x \leftarrow m\}$ / A ∃x.B	

This leads to the following $\Box$, $\Diamond$, and $\boxtimes$, $\diamond$ properties:

(1) ***vld*** $(\Box(A \wedge B) \Leftrightarrow \Box A \wedge \Box B)$
(2) ***vld*** $(\boxtimes(A \wedge B) \Leftrightarrow \boxtimes A \wedge \boxtimes B)$
(3) ***vld*** $(\Box(A \vee B) \Leftrightarrow \Box A \vee \Box B)$
(4) ***vld*** $(\boxtimes(A \vee B) \Leftrightarrow \boxtimes A \vee \boxtimes B)$

In the following table, it is propositioned that $\Box$, $\Diamond$, and $\boxtimes$, $\diamond$ are modal S4:

$(\Diamond)$	$/\mathbf{T}\ \Diamond A \Leftrightarrow \neg\Box\neg A$	$(\boxtimes)$	$/\mathbf{T}\ \boxtimes A \Leftrightarrow \neg\boxtimes\neg A$
$(\Box K)$	$/\mathbf{T}\ \Box(A \Rightarrow B) \Rightarrow (\Box A\, \Box B)$	$(\boxtimes K)$	$/\mathbf{T}\ \boxtimes(A \Rightarrow B) \Rightarrow (\boxtimes A \Rightarrow \boxtimes B)$
$(\Box T)$	$/\mathbf{T}\ \Box A \Rightarrow A$	$(\boxtimes T)$	$/\mathbf{T}\ \boxtimes A \Rightarrow A$
$(\Box 4)$	$/\mathbf{T}\ \Box A \Rightarrow \Box\Box A$	$(\boxtimes 4)$	$/\mathbf{T}\ \boxtimes A \Rightarrow \boxtimes\boxtimes A$
$(\Box M)$	$A\ B / \Box A\ \Box B$	$(\boxtimes M)$	$A\ B / \boxtimes A\ \boxtimes B$
$(\Box \wedge)$	$\Box(A \wedge C)\ B\ //\ \Box A \wedge \Box C\ B$	$(\boxtimes \wedge)$	$\boxtimes(A \wedge C)\ B\ //\ \boxtimes A \wedge \boxtimes C\ B$
$(\Box \vee)$	$A\ \Box(C \vee B)\ //\ A\ \Box C \vee \Box B$	$(\boxtimes \vee)$	$A\ \boxtimes(C \vee B)\ //\ A\ \boxtimes C \vee \boxtimes B$

It is noted, that because

$\neg$***vld*** $\Diamond A \Rightarrow \Box\Diamond A$
$\neg$***vld*** $\boxtimes A \Rightarrow \boxtimes\diamond A$

the modalities are not S5.

Finally, location properties, location rules, composition properties, and composition rules are listed.

Location Properties
(1) ***vld*** $(n[A \wedge B] \Leftrightarrow n[A] \wedge n[B])$
(2) ***vld*** $(n[A \vee B] \Leftrightarrow n[A] \vee n[B])$

Location Rules

$(n[])$	$A\ B\ //\ n[A]\ n[B]$
$(n[]\wedge)$	$n[A \wedge C]\ B\ //\ n[A] \wedge n[C]\ B$
$(n[]\wedge)$	$A\ n[C \vee B]\ //\ A\ n[C] \vee n[B]$

Composition Properties

(1) ***vld*** $(A \mid B \Rightarrow B \mid A)$
(2) ***vld*** $(A \mid (B \mid C) \Leftrightarrow (A \mid B) \mid C)$
(3) ***vld*** $((A \wedge B) \mid C \Rightarrow A \mid C \wedge B \mid C)$
(4) ***vld*** $((A \vee B) \mid C \Rightarrow A \mid C \vee B \mid C)$

Composition Rules

	(|)	A' B'; A" B" / A' | A" B' | B"
5	(|∧)	(A∧B) | C D / A | C ∧ B | C D
	(|∨)	A (B∨C) | D / A B | D ∨ C | D
	(|Π)	/ A' | A" ∧ B' Π B" A' | B" ∨ B' | A"
	(|¬)	/ ¬(A' | A") ∧ ¬(B' | B") ¬(B' | A") ∨ (¬A' | ¬B")
	(|-E)	A B' | B"; A'∧(B' | C") D; A"∧(C' | B") D
10		/ (A∧(A'∧A"))∧(C' Π C") D

Adjunctions

The following propositions and corollaries relate to location adjunct rules, and composition adjunct rules. The first proposition states that A@*n* and *n*[A] are adjuncts.

Proposition: Location Adjunct Rules

(*n*[] @) *n*[A] B // A B@*n*

Corollaries

(1) ***vld*** *n*[A@*n*] ⇒ A

(2) ***vld*** A ⇒ *n*[A]@*n*

Proposition: Composition Adjunct Rules

(|>) A | C B // A C>B

Corollaries

(1) ***vld*** A >B | B ⇒ B

(2) ***vld*** A ⇒ B>(A B)

(3) ***vld*** A>B | B>C ⇒ A>C

Reflecting Validity

In this sub-section, validity and satisfiability are reflected into the logic, by means of the > operator:

Vld A ≅ (¬A)>**F**

Sat A ≅ ¬(A>**F**)

From this validity and satisfiability, two propositions and one lemma are described:

Proposition: *Vld* and *Sat*

(1) ***vld*** *Vld* A ⇔ ***vld*** A
(2) ***vld*** *Sat* A ⇔ ***sat*** A

Lemma: *Vld, Sat* Properties

(1) ***vld*** (*Vld*(A∧B) ⇔ *Vld*A ∧ *Vld*B)
(2) ***vld*** (*Vld*(A∨B) ⇔ *Vld*A ∨ *Vld*B)

Proposition: *Vld, Sat* is Modal S5

(*Sat*)	/ **T** *Sat*A ⇔ ¬*Vld*¬A
(*Vld* K)	/ **T** *Vld*(A ⇒ B) ⇒ ((*Vld*A) ⇒ (*Vld*B))
(*Vld* T)	/ **T** (*Vld*A) ⇒ A
(*Vld* 5)	/ **T** (*Sat*A) ⇒ (*Vld Sat* A)
(*Vld* M)	A B / *Vld*A *Vld*B
(*Vld* ∧)	*Vld*(A∧C) B // *Vld*A ∧ *Vld*C B
(*Vld* ∨)	A *Vld*(C∨B) // A *Vld*C ∨ *Vld*B

Reflecting Name Equality

Finally, it is noted that it is possible to encode name equality within the logic in terms of validity. It is recalled that *an* $n \cong n[\mathbf{T}] \mid \mathbf{T}$. One proposition then follows.

~~$m = n \cong Vld$(an m > an n)~~

Proposition

vld $m = n$ ⇔ the names m and n are equal

Examples

In this section of the detailed description, examples of mobile computing environments in conjunction with the modal logic of the preceding section are presented. Specifically, four separate situations are shown in the diagram of FIG. 6, and an additional situation is shown in the diagram of FIG. 7. Those of ordinary skill within the art can appreciate that the situations of FIGs. 6 and 7 are examples for illustrative purposes only, and do not represent a limitation on the invention.

Referring first to FIG. 6, four situations are presented, situations 600, 602, 604 and 606. In situation 600, a container *n* includes a process *Q*, and includes a policy telling the container how to behave. Specifically, the policy is *in m.P*, which instructs the container *n* including the process *Q* to move into the container *m* already having the policy *R* therein, as shown in situation 600. In situation 602, a container *n* includes a process *Q*, and the policy telling the container how to behave is *out m.P*, which instructs the container *n* including the process *Q* to move out of the container *m* also having the policy *R* therein, as shown. In situation 604, the policy or instruction *open n.P* is executed on the container *n* having the process *Q*, such that *Q* exits the container *n* as a result. Finally, in situation 606, a replicated instruction is executed on the process *P*, such that an additional process *P* is made (that is, process *P* is copied).

Referring next to FIG. 7, a communication operation referred to as a note is shown in the situation 700. The note can reside within a container. The capabilities that can be held by the note include names, such as *n*, as well as action capabilities, such as *in n*, *out n*, *open n*, or a path, such as *C.C'*, as has been described in the modal logic section of the detailed description.

Methods

In this section of the detailed description, computer-implemented methods according to varying embodiments of the invention are presented. The methods make use of the modal logics described in the previous section of the detailed description, which are based on ambient calculus and provide for spatial relationships among processes of containers. The methods relate to a model-checking algorithm. The computer-implemented methods are desirably realized at least in part as one or more programs running on a computer -- that is, as a program executed from a computer- or machine-readable medium such as a memory by a processor of a computer. The programs are desirably storable on a machine-readable medium such as a floppy disk or a CD-ROM, for distribution and installation and execution on another computer.

As described herein, the method references sub-methods norm, sublocation and reachable. In one embodiment of the invention, these sub-methods are implemented as described in a succeeding embodiment of the invention.

Referring now to FIG. 3, a flowchart of a method according to an embodiment of the invention is shown. In 300, a process is input. This is the process that is to be analyzed. The process may be a thread, an applet, an agent, etc.; the invention is not so limited. The process itself may be a composition of one or more processes. For example, the process can be the composition $P|Q|R$, where each of P, Q and R is a separate process. Again, the invention is not so limited.

302, 304, 306, 308, 310, 312, and 314 implement the analysis of the process against a formula, using a predetermined modal logic based on ambient calculus,

according to one embodiment of the invention. The formula against which the process is to be analyzed can be a policy, such as a security policy or a mobility policy, such that the policy is described using the predetermined modal logic, such as has been described in the preceding sections of the detailed description. In one embodiment, the process is analyzed in a recursive manner. The analysis of 302, 304, 306, 308, 310, 312 and 314 can be summarized as a theorem, specifically, *for all restriction-free process P and >-free closed formulas A, P A if and only if Check(P,A), where Check() is the analysis of 302, 304, 306, 308, 310, 312 and 314.*

In 302 specifically, the process is analyzed in three ways, referred herein as an initial checking of the process against the formula. First, it is checked that $Check(P,\mathbf{T}) = \mathbf{T}$. This means that if the formula is **T** then the outcome of the analysis is **T** for any process. Second, it is checked that $Check(P,\neg A) = \neg Check(P,A)$. This means that if the formula is a negation $\neg A$ then the outcome of the analysis is the negation of a recursive analysis of the process *P* against formula A.. Third, it is checked that $Check(P,A \vee B) = Check(P,A) \vee Check(P,B)$. This means that the outcome of the analysis is the disjunction of recursively checking the process *P* against formula A and checking the process *Q* against formula B.

In 304 specifically, the process is normalized, and it is determined whether the process includes only one element, or entry. If there is more than one element, then the process fails against the policy. The check of 304 only applies if the formula is a location *n*[A]. This check can be expressed as:

$$Check(P, n[A]) = \begin{cases} Check(Q,A) & \text{if } Norm(P) = [n[Q]] \text{ for some } Q \\ \mathbf{F} & \text{otherwise} \end{cases}$$

In 306 specifically, the process is partitioned to determine whether each component of the process satisfies the formula, or policy. If any component fails against the policy, then the process itself fails. The check of 306 only applies if the formula is a composition A ' B. This check can be expressed as:

$$Check(P, A \mid B) = \text{let}[\pi_1, ..., \pi_k] = Norm(P) \text{ in}$$
$$\begin{cases} \mathbf{T} \text{ if } \exists I, J . I \cup J = 1..k \wedge I \cap J = \phi \wedge \\ \qquad Check(\prod_{i \in I} \pi_i, A) \wedge Check(\prod_{j \in J} \pi_j, B) \\ \mathbf{F} \quad \text{otherwise} \end{cases}$$

In 308 specifically, all of the names of the process are determined. Then, it is verified that a name exists for the formula that is unequal to any of the names of the process. If this verification fails, then the process itself fails against the policy. The check of 308 only applies if the formula is an existential quantification $\exists x.A$. This check can be expressed as:

$$Check(P, \exists x.A) = \text{let}\{m_1, ..., m_k\} = fn(P) \cup fn(A) \text{ in}$$
$$\text{let } m_0 \notin \{m_1, ..., m_k\} \text{ be some fresh name in}$$
$$\begin{cases} \mathbf{T} \text{ if } Check(P, A\{x \leftarrow m_i\}) \text{ for some } i \in 0..k \\ \mathbf{F} \text{ otherwise} \end{cases}$$

In one embodiment, a unification algorithm, as known within the art, can be used to effectuate the check of 308, to make the check more efficient. However, the invention is not so limited.

In 310 specifically, each sublocation of the process is checked, or analyzed, against the formula, or process. If the check fails for any sublocation, then the process itself fails against the policy. The check of 310 only applies if the formula is a

somewhere modality ✧A. This check can be expressed as:

$$Check(P,\text{✧A}) = \text{let}[P_1,...,P_k] = SubLocations(P)\text{ in}$$
$$\begin{cases}\mathbf{T}\text{ if } Check(P_i,\text{A})\text{ for some } i \in 1..k \\ \mathbf{F}\text{ otherwise}\end{cases}.$$

In 312 specifically, the spatial reach of the processed is checked, or analyzed, against the formula, or process. This check thus determines whether the process has a finite spatial reach. If the check fails, then the process itself fails against the policy. The check of 312 only applies if the formula is a sometime modality ◊A. This check can be expressed as:

$$Check(P,,\text{A}) = \text{let}[P_1,...,P_k] = Reachable(P)\text{ in}$$
$$\begin{cases}\mathbf{T}\text{ if } Check(P_i,\text{A})\text{ for some } i \in 1..k \\ \mathbf{F}\text{ otherwise}\end{cases}.$$

In 314 specifically, it is checked recursively that the process satisfies a formula when enclosed in a surrounding ambient. If the check fails, then the process itself fails against the policy. The check of 314 only applies if the formula is a location adjunct A@*n*. This can be expressed as $Check(P,\text{A}\ @n) = Check(n[P],\text{A})$.

Finally, in 316, whether or not the process satisfied the formula – based on the analysis conducted in 302, 304, 306, 308, 310 and 312, is output. The invention is not limited to the manner by which output is accomplished. For example, in one embodiment, it can be output to a further analysis program or software component, that allows for analysis and conclusions to be drawn. As another example, the output can be displayed on a display device, or printed to a printer, etc. As a third example, output can mean storage to a storage device, for later and/or further analysis by a program or software component.

As can be appreciated by those of ordinary skill within the art, the above method can be effectuated by a system in one embodiment of the invention. That is, a system including a processor and a computer-readable medium, such that first data stored on the medium represents the process, and second data stored on the medium represents the formula. In such an instance, an analysis program is executed by the processor from the medium to analyze the process against the formula, for example, in a recursive manner.

Sub-Methods

In this section of the detailed description, the sub-methods norm, reachable, and sublocations, as referenced in the previous section of the detailed description, are described, according to one embodiment of the invention. However, the invention is not so limited to the embodiment of this section.

First, the sub-method norm is described. Any replication-free process may be factored up to structural congruence into a normal form consisting of a composition of prime processes, where a prime process is an ambient, an action, an input, or an output.

In the following table, the prime processes are first defined. The normal form is stated in terms of the following notation: for processes $P_1,...,P_k$, let the notation $\prod_{i\in 1..k} P_i$ be short for the composition $P_1 \mid ... \mid P_k \mid 0$.

Prime processes, and normal forms:

$\pi ::=$	Prime process
$M[P]$	Ambient
$M.P$ for $M \in \{in\ N, out\ N, open\ N, n\}$	Action
$(x).P$	Input
$\langle M \rangle$	Ouput

Next, an algorithm is defined for computing normal forms. Given a process, the following function returns a list of primes, which represents a normal form of the process. The notation $[\pi_1,...,\pi_k]$ is used for a list of primes. List concatenation is written as follows: $[\pi_1,...,\pi_k]++[\pi'_1,...,\pi'_l]=[\pi_1,...,\pi_k,\pi'_1,...,\pi'_1]$. Then the notation $P\in[P_1,...,P_k]$ is used as a shorthand for $P\in\{P_1,...,P_k\}$.

Computing a normal form of a replication-free process:

$$
\begin{aligned}
&Norm(M[P]) = [M[P]]\\
&Norm(0) = [\ ]\\
&Norm(P\mid Q) = Norm(P)++Norm(Q)\\
&Norm(M.P) = Norm(P) && \text{if } Head(M)=\in\\
&Norm(M.P) = [M_1.(M_2.P)] && \text{if } Head(M) = M_1.M_2\\
&Norm((x).P) = [(x).P]\\
&Norm(\langle M\rangle) = [\langle M\rangle]
\end{aligned}
$$

Since all the recursive calls are on subprocesses of the original process, the algorithm always terminates. Moreover, *if* $Norm(P)=[\pi_1,....,\pi_k]$ *then* $P\equiv\prod_{i\in 1..k}\pi_i$.

Next, the sub-method sublocations is described. An algorithmic characterization of the $P\downarrow^{\bullet}P'$ predicate is used, which is used in the definition of the spatial modality. Specifically, we define a procedure *SubLocations*(*P*) for computing representatives of all processes P' such that $P\downarrow^{\bullet}P'$. The definition of *SubLocations*(*P*) depends on a subroutine *Children*(*P*), which computes representatives of all processes P' such that $P\downarrow P'$.

Computing the children of a normal form is as follows:

$$Children([\,]) = [\,]$$

$$Children(P :: Ps) = \begin{cases} Q :: Children(Ps) & \text{if } P = n[Q] \\ Children(Ps) & \text{otherwise} \end{cases}$$

The following lemma and proposition are then given as:

Lemma *Suppose* $Children([\pi_1,...,\pi_l]) = [P_1,...,P_k]$.

(1) *For all* $i \in 1..k, \prod_{j \in 1..l} \pi_j \downarrow P_i$.

(2) *If* $\prod_{j \in 1..l} \pi_j \downarrow Q$ *then* $Q \equiv P_i$ *for some* $i \in 1..k$.

Proposition *Suppose* $Children(Norm(P)) = [P_1,...,P_k]$.

(1) *For all* $i \in 1..k, P \downarrow P_i$.

(2) *If* $P \downarrow Q$ *then* $Q \equiv P_i$ *for some* $i \in 1..k$.

Computing the sublocations of a process is then given as:

$$SubLocations(P) = \text{let}[P_1,...,P_k] = Children(Norm(P)) \text{ in } [P] + +SubLocations(P_1) + +... + +SubLocations(P_k)$$

The following lemma is needed, however. Note that it cannot be generalized to the reflexive case, that is, where $\downarrow^*$ is substituted for $\downarrow^+$.

Lemma *If* $P' \equiv P, P \downarrow^+ Q$, *and* $Q \equiv Q'$, *then* $P' \downarrow^+ Q'$.

A proposition is next given as,

Proposition *Suppose* $SubLocations(P) = [P_1,...,P_k]$.

(1) *For all* $i \in 1..k, P \downarrow^* P_i$.

(2) *If* $P \downarrow^* Q$ *then* $Q \equiv P_i$ *for some* $i \in 1..k$.

Finally, the sub-method reachable is described. Computing the sublocations of a process gives:

$$Reachable(P) = \text{ let } [P_1,...,P_k] = Next(P) \text{ in } [P] + +Reachable(P_1) + +... + +Reachable(P_k)$$

There is one lemma and one proposition associated with this,

Lemma *If* $P' \equiv P, P \rightarrow^+ Q$, *and* $Q \equiv Q'$, *then* $P' \rightarrow^+ Q'$.

Proposition *Suppose* $Reachable(P) = [P_1,...,P_k]$.

(1) *For all* $i \in 1..k, P \rightarrow^* P_i$.

(2) *If* $P \rightarrow^* Q$ *then* $Q \equiv P_i$ *for some* $i \in 1..k$.

Conclusion

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is manifestly intended that this invention be limited only by the following claims and equivalents thereof.